MiniBooNE Data Collection

Analysis

Fri, Feb 1, 2002

In order to monitor MiniBooNE accelerator operation, time stamped measurements of beam profiles and other data need to be collected for each Main Injector ramp cycle of interest. To that end, the front-end will log such data for multiple cycles so that the Acnet console has a chance to retrieve it. MiniBooNE cycles are 15 Hz cycles, although they may not always be consecutive 15 Hz cycles. It is expected that they might occur at an average rate of as much as 5 Hz. A console that needs to monitor such data will have to collect it at an average rate that is fast enough to keep up. If a console made data requests for 1 Hz data, for example, the front-end would need to have at least that much data buffered up for delivery to the console.

In order not to wed the front-end logic too closely with the application's plans for displaying and analyzing the data, let us assume that the console needs to request the data by MI reset at a rate that is sufficient to keep up, but that the front-end does not try to accumulate a complete MI cycle of data before making it available to the console application. Let the application organize the data it collects in whatever way it sees fit. The front-end should only make the data available to the application in a timely manner.

Along with each set of measured data, there will be timing information sufficient to permit the application to build up correlated data of interest. Each set of data will be tagged as belonging to a given MI reset event and number of 15 Hz cycles following the MI reset event. (Even when MI events are accumulated in separate FE buffers, the data should be tagged with the reset event, because one of the buffer types includes all MI reset events.)

By collecting the FE data at whatever reasonable rate the application desires, say from 1 Hz to 5 Hz, the application can display more ongoing data even as the MI cycle proceeds, in order to give a better feel to the operator for the real-time accelerator activity. Note that questions of how many 15 Hz cycles to collect per MI reset can be decided at the console level; the FE does not need to be concerned about it.

The model for buffering can be taken from that provided by data stream support in the IRM front ends. For each MiniBooNE cycle, defined as a Booster `0x1D` event, measured BPM data and other values can be buffered as a record of fixed size written into a data stream of suitable size to hold at least a few seconds of data records. The structure of this record must be defined to include the time stamp data as well as the measured beam related data. There must be at least (`N+1`) data streams defined, where `N` is the maximum number of MI reset ramp cycles that exist, at least those relevant for MiniBooNE. Within one FE all such MiniBooNE data can be recorded in a single record structure. In this way, the time stamp information occurs only once for the data records obtained from each front end. (If it is more convenient, a separate data stream could be used for each beam related device within a FE, but each such data stream record would need the extra overhead of time stamp data in order to allow correlation of the received data at the console.)

The console application would make requests for all the relevant devices it needs, and data records will flow from each FE, not necessarily synchronously, to the console. The application, as it sees the records arrive, should copy it into its own data structures for subsequent processing. When it has a set of suitably correlated values, it can see that the results are displayed appropriately. If it wants to maintain copies of correlated data for the last several MI cycles of particular interest, it can do that. Again, that part of the logic is not a concern for the front end. The FE provides the data; the application disposes of it.

Note that different applications acting on the same data simultaneously, but in different ways, encounter no conflict here. The delivery of the data to a console does not consume the data, so that it is still available for other clients. Requests from different consoles, say at 1 Hz, will not necessarily be synchronously delivered at all. But each application in this scenario will receive all of the data it desires, whether it matches another application's interest or not.

To review the logic behind data stream support in an IRM, the data records are written by a local application that is designed to collect the right stuff into a record and write it to a suitable data stream. (It will probably write the same record into two data streams, since one data stream is to contain data relevant for any MI reset.) Once a record is written, it is recorded and is available to be delivered to a console in a reply buffer of length specified by the application, at a rate specified by the application. The size of data requested should be enough to contain a maximum number of records plus 4 bytes to allow for a header that includes a count of the number of records that the application will find in the reply buffer.

Consider an example, where the size of each data stream record is 100 bytes. If the application makes a 1 Hz request, it might ask for 604 bytes, for example. This specifies a reply buffer size sufficient to hold a maximum of 6 records plus the 4-byte header that includes the record count. That counter value can range from 0–6, in this example. Note that in this case, the application assumes that the average rate of record logging in the data stream is not more than 6 records per second. The FE only delivers replies at the rate requested and of the size requested; it is the application's responsibility to ensure that it is fast enough.

There are several choices for an application to request data stream records, according to the contents of the `SSDN` field relevant to the Acnet reading property. For IRMs, the parameter is called a listype#. For listype 50 (decimal), the request specifies that data replies start when the request is made. It is suitable for delivery of multiple replies but not for one shots. (A one shot request will likely yield no records in the reply buffer.) For listype 51, a one shot request will fill the reply buffer with as many records as fit, assuming that they exist within the data stream queue, and with the last record the most recent. If repetitive replies are requested, the first one will have the same selection of records as for the one shot case, with subsequent replies including whatever records are recorded as time goes on. For listype 78, the starting point will be the oldest record in the queue for building the first reply, or the only reply in case of a one shot request. Subsequent replies will move forward from there. This case allows for reading out the entire set of data already recorded and available in the queue and then to keep up with the new records written as time goes on. Of course, it is the responsibility of the requester to specify a suitable reply rate and buffer size to keep up with the ongoing new data records. For each reply, the front end remembers where the requester left off, so that the next reply continues from that point. This memory is kept with the request, not with the data stream queue, so that multiple requests have separate such memories of where each left off.

It is important that the size of each record be known in the above scenarios. The application receives a structure that includes a 4-byte header followed by an array of records. The data stream is defined, and the corresponding queue is initialized, at front end reset time. An alternative is also supported that permits the data stream records to be of variable length. In order to permit processing of the reply data, each record is preceded by its size. This may complicate processing of the data by the application enough that the fixed record size approach is preferred. Obviously, the record size and the size of the queue should be chosen with care according to the uses to be put to the data by the various applications that will use it. In addition, the detailed structure of the records must be known equally well by the front end that records it and the application that interprets it.

An alternative approach to periodic replies of MiniBooNE data is clock event based replies. Since event `0x1D` is used for announcing MiniBooNE cycles, a request can be based upon `0x1D` events, so that replies occur only on cycles in which MiniBooNE data has been collected. But this approach is unreliable, because such cycles may occur on consecutive 15 Hz cycles. `RETDAT` replies cannot reliably be received by Acnet consoles if they occur one 15 Hz cycle apart, since console application activity is not synchronized with accelerator operation. The fastest rate of reliable collection of replies to `RETDAT` requests is 7.5 Hz, in which the application queries the data pool for reply data at its nominal 15 Hz rate. In recognition of this, one might imagine collecting data on every other `0x1D` event, but there is no way to specify this return logic in a `RETDAT` request.

Consider returning data on MI reset events. In this case, the occurrence of the reset event is well before the start of MiniBooNE cycles. If it were desired to collect MiniBooNE data on `0x29` events, for example, it means collecting data from all MiniBooNE cycles following that event until the MI ramp completes, or perhaps when the next MI reset event occurs. But the next MI reset event may be any MI reset event, so how can the application know which MI reset event to specify that will come after the sequence of MiniBooNE cycles of interest? An exception might be for the case in which the application wants data from all MI resets; then one could specify the any-MI reset event. But there is no way to do that via `RETDAT`. (One can do this via `FTPMAN`.)

One might consider requesting data for return on a specified MI reset event, anticipating receiving the data from the previous such MI ramp cycle. But for infrequent cases, this could interpose an unacceptably long delay before the data was delivered. It appears there is no way to avoid requesting data via 1 Hz replies, even when there may be no such data present in some of the replies.